Wrocław, 18 marca 2024

Paradygmaty Programowania Obiektowego

Laboratorium 1

Enkapsulacja

prowadzący: *mgr inż. Marta Lampasiak*

1 Wprowadzenie

Głównym celem zajęć jest implementacja programu wykorzystującego mechanizmy enkapsulacji. Jednocześnie podczas zajęć zostaną poruszone takie zagadnienia, jak: klasy, obiekty, lista inicjalizacyjna, funkcje składowe, funkcje modyfikujące i dostępowe, konstruktor, destruktor.

Klasa to inaczej mówiąc **typ**, ponieważ kreujemy nie jeden, konkretny obiekt, a **klasę** obiektów. Klasa składa się ze słowa kluczowego *class*, po którym występuje jej nazwa. Następnie w klamrach umieszczamy tzw. **ciało** klasy, czyli określenie z czego się składa. Posiada ona składowe w postaci danych (cech obiektów) oraz tzw. **funkcje składowe** (nazywane często **metodami**).

Klasa jest jak kapsuła, w której zamknięto dane oraz funkcje do posługiwania się nimi. Ta czynność zamknięcia nazywa się enkapsulacją („kapsułowaniem"). Jest to ważna cecha języka C++, ponieważ odzwierciedla nasze myślenie o obiektach. Obiekt typu czajnik to nie tylko jego składowe, czyli np. grzałka, ale też akcja jaką wykonuje, przykładowo funkcja *gotuj*.

Jeżeli klasa to nasza kapsuła, to pojęcie hermetyzacji można by metaforycznie rozumieć jako to, czy kapsuła jest zbudowana z przezroczystego materiału czy nie. Innymi słowy, czy coś może być dostępne spoza klasy czy nie. Wszystkie pola klasy są widoczne wewnątrz klasy. Oznacza to, że mają do nich dostęp funkcje składowe danej klasy. Mogą one jednak mieć różny poziom dostępności z zewnątrz, a więc z funkcji które nie są składowymi danej klasy. **Poziom dostępności** jest określany jednym ze **słów kluczowych**: *public*, *private*, lub *protected*.

Poniżej znajduje się fragment kodu, który pokazuje definicję przykładowej klasy o nazwie *Osoba*. Zauważ, że funkcja składowa klasy ma dostęp do jej pól prywatnych i nie muszą być one przekazywane jako argumenty. W kodzie przedstawiono również utworzenie obiektu klasy.

```
#include <iostream>
using namespace std;

// ----- poczatek definicji klasy -----
class Osoba
{
private:  // skladniki prywatne
  string nazwisko_;
  int wiek_;

public:   // skladniki publiczne

  void zapamietaj(string, int); // deklaracja funkcji skladowej (metody) klasy

  // definicja funkcji skladowej w ciele klasy
  void wypisz()
  {
    cout << nazwisko_ << ", lat: " << wiek_ << endl;
  }
};
// ----- koniec definicji klasy -----


```

```
24 // definicja funkcji skladowej (metody) poza cialem klasy
25 // jest poza klasa, a wiec nazwa funkcji zostaje uzupelniona
26 // nazwa klasy i operatorem zakresu, czyli ::
27 void Osoba::zapamietaj(string napis, int lata)
28 {
29   nazwisko_ = napis;
30   wiek_ = lata;
31 }
32
33 int main()
34 {
35   Osoba osoba1; // utworzenie obiektu klasy
36   osoba1.zapamietaj("Potter", 7); // wywolanie metody dla obiektu klasy
37   osoba1.wypisz(); // wywolanie metody dla obiektu klasy
38   return 0;
39 }
```

Kolejnym zagadnieniem poruszanym na laboratorium jest specjalna funkcja składowa nazywana **konstruktorem**. Konstruktor charakteryzuje się tym, że nazywa się tak samo jak klasa. Nie posiada on żadnego określonego typu (nawet *void*). Może być **domyślny** (domniemany), czyli taki, który zostaje wywołany z pustą listą argumentów, ale mogą zostać utworzone również innego rodzaju konstruktory. Należy jednak pamiętać, że gdy tworzony jest obiekt danej klasy, wywoływana jest jedna implementacja konstruktora, którą kompilator może dopasować do listy parametrów inicjalizacyjnych obiektu, zgodnie z regułami przeciążenia. Jeżeli kompilator nie byłby w stanie dopasować żadnej definicji konstruktora, zgłoszony zostanie błąd w czasie kompilacji.

Mówiąc o konstruktorze warto wspomnieć o tzw. **liście inicjalizacyjnej**. Nie jest ona konstruktorem, ale pewnym mechanizmem pozwalającym na przypisanie wartości parametrom przed stworzeniem obiektu. Może być przydatna gdy obiekt przechowuje referencje, które muszą być zainicjalizowane przed powstaniem obiektu.

Specjalną funkcję składową klasy stanowi również **destruktor**. Wywoływany jest on automatycznie przed usunięciem obiektu z pamięci. Jego działanie jest zatem w pewnym sensie przeciwstawne do działania konstruktora. Destruktor nie może ulegać przeciążeniom.

2 Zadania

Zadanie opiera się na utworzeniu klasy o nazwie `Gracz`, implementacji odpowiednich metod oraz funkcji działających na obiektach, a także na wyciągnięciu odpowiednich wniosków z przeprowadzonych testów. Na pytania zadane w poleceniach należy odpowiadać w komentarzach (odpowiedzi należy uzasadnić). Stanowią one część oceny. Poniżej znajduje się lista poszczególnych kroków, jakie należy wykonać wraz z wyszczególnieniem progów ocen. W celu uzyskania maksymalnej oceny należy wykonać wszystkie kroki.

1. Polecenia na ocenę 3.0:
   (a) Utwórz klasę o nazwie `Gracz` zawierającą pola **publiczne**:
      - `nick_` – zmienna typu `string`, może być to napis kilkuczłonowy,
      - `level_` – zmienna typu `int`,
      - `polozenie_x_` – zmienna typu `float`,
      - `polozenie_y_` – zmienna typu `float`.

   (b) Napisz **funkcję** o nazwie `wypisz`, która jako argument otrzymuje obiekt tej klasy i wypisuje na standardowym wyjściu wartości pól otrzymanego w argumencie obiektu.

   (c) Napisz **funkcję** `wczytaj`, która wczytuje ze standardowego wejścia wartości pól do obiektu, który podano w jej argumencie. Zauważ, że może wystąpić potrzeba wczytania napisu kilkuczłonowego, czyli razem ze spacją. Przygotuj swój program na tego rodzaju przypadek.

(d) Dopisz do klasy **metody**: wczytującą (ze standardowego wejścia dane dla pól obiektu) i wypisującą na standardowe wyjście pola obiektu, aby się nie pomylić najlepiej nazwij metody inaczej niż wcześniej stworzone funkcje.

(e) Zaprezentuj działanie wszystkich utworzonych funkcji.

(f) Zmień dostęp tylko do pól klasy z publicznego na prywatny. Funkcje składowe (metody) niech pozostaną publiczne. Przetestuj i odpowiedz w komentarzu w kodzie. PYTANIE: Czy stworzone przez Ciebie funkcje składowe i funkcje nie będące składowymi klasy będą nadal działać? Jeśli tak, to które i dlaczego? (Niedziałające elementy zakomentuj)

(g) Utwórz konstruktor domyślny, który będzie nadawał początkowe wartości oraz wyświetli napis: *Nadanie wartosci – konstruktor domyslny.* Następnie utwórz obiekt i wyświetl jego dane, czyli zaprezentuj działanie konstruktora domyślnego.

(h) Utwórz destruktor, który w swoim ciele jedynie wyświetla na standardowym wyjściu tekst: *Likwiduje!.*

(i) PYTANIE: Czy dla stworzonego obiektu zostały wyświetlone wartości, które nadałeś poprzez konstruktor?

(j) PYTANIE: Czy zauważyłeś działanie destruktora? Czy musiałeś go wywołać jawnie?

2. Polecenia na ocenę 4.0:

(a) Utwórz konstruktor wieloargumentowy (parametryczny), który podczas tworzenia obiektu nada polom klasy wartości przekazane w argumentach. Wykorzystaj **listę inicjalizacyjną**. Definicję konstruktora umieść poza klasą. Dodatkowo w konstruktorze wyświetl na standardowym wyjściu tekst: *Nadanie wartosci – konstruktor wieloargumentowy.*

(b) Utwórz obiekt korzystając z tego konstruktora.

(c) PYTANIE: Przetestuj i odpowiedz, w jakiej kolejności wywołają się destruktory w przypadku utworzenia obiektu za pomocą konstruktora domyślnego, a potem kolejnego za pomocą konstruktora wieloargumentowego?

3. Polecenia na ocenę 5.0:

(a) W klasie możemy zdefiniować tzw. funkcje modyfikujące i funkcje dostępowe. Te drugie powinny zostać zdefiniowane z pewnym słowem kluczowym. Umieść je przy odpowiednich, stworzonych w tym zadaniu przez siebie funkcjach.

(b) PYTANIE: Wyjaśnij, dlaczego akurat przy tych funkcjach użyłeś słowa kluczowego (dodatkowo podaj jakie to słowo)?

(c) PYTANIE: Jeżeli klasa posiada konstruktor wieloargumentowy, to konieczne było zdefiniowanie konstruktora domyślnego **jawnie**, aby mógł on zostać wywołany? Czy gdybyś tego nie zrobił, zostałby wywołany niejawnie? (Przetestuj i zastanów się, dlaczego)